(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 069 716 A2

(12) # EUROPEAN PATENT APPLICATION

(43) Date of publication:
17.01.2001 Bulletin 2001/03

(51) Int Cl.7: **H04H 7/00**, H04N 5/262

(21) Application number: **00305883.1**

(22) Date of filing: **12.07.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **14.07.1999 US 352089**

(71) Applicant: **GENERAL ELECTRIC COMPANY**
Schenectady, NY 12345 (US)

(72) Inventor: **Kenny, Kevin Bernard**
Niskayuna, New York 12309 (US)

(74) Representative: **Goode, Ian Roy et al**
GE LONDON PATENT OPERATION,
Essex House,
12/13 Essex Street
London WC2R 3AA (GB)

(54) **A throttler for rapid start-up of a program controlled broadcast system**

(57) A throttler method for rapid start-up for use with broadcast automation systems. A throttler (100) loads an initial playlist (106) while also accepting editing commands (108). The throttler interleaves these events and commands and generates and modifies the playlist of scheduled events. The throttler sends the events to a broadcast automation system (118) for execution which drives audio and video devices (120) based on the scheduled events, allowing the editing commands (108) to be interleaved with non-editing commands (114). For unprocessed editing command, a command pair of up to two pieces of information are maintained: one deletion command (110) and one insertion command (112). Each command, or event, has a unique "event identifier" and is hashed into a rapidly accessible priority queue table, according to urgency.

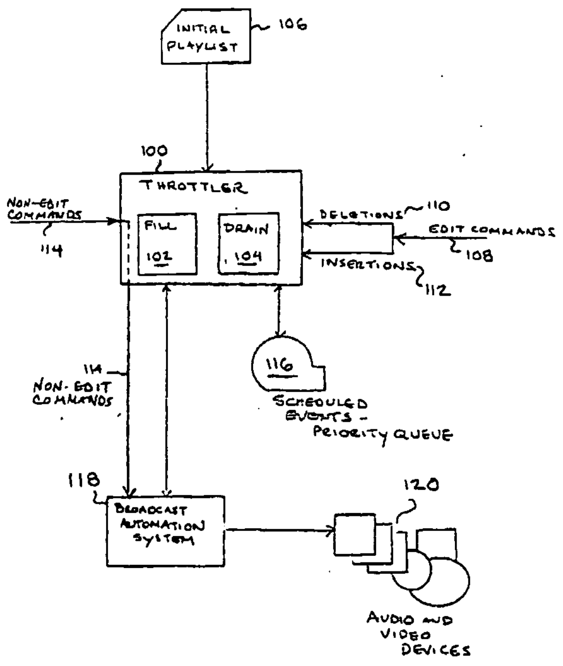


FIG. 1

EP 1 069 716 A2

Description

[0001] This invention relates to broadcast automation systems, and more particularly to a method for rapid start-up for these systems.

[0002] Present-day broadcast automation systems generally work on the concept of a "playlist", also known as a schedule of events. These events are commands to video devices to play pieces of audio/visual material, insert special effects, acquire video from a particular input device, direct video to particular output devices, and other activities related to audio/video broadcasting.

[0003] Broadcast automation systems operate by loading the events of an entire playlist sequentially, all at once. While the playlist is loading, the system is unavailable for other processing while this initial playlist is loading. While the system can subsequently accept changes, called "edits" to the playlist, the processing of edits is limited. A large number of edits in rapid succession can make the systems unavailable while the edits are being processed. Moreover, edits to events that will not occur until far in the future, for instance, appending additional material to the playlist, can indefinitely delay edits to events that will occur sooner. This can result in lost edits or erroneous execution of the playlist.

[0004] In an exemplary embodiment of the invention, a software component called a "throttler" allows playlist loads and edits to be interleaved with other actions such as sending commands to devices and interacting with an operator. External components that load and edit the playlist send editing commands. Each command represents either an insertion or a deletion of an event. Modification to an existing event is expressed as a deletion of the existing event, followed by an insertion of the modified event. Every event has a unique "event identifier" which points to a rapidly accessible data structure of command pairs of insertion and deletion edits for that event, ordered by urgency.

[0005] The interleaving of commands has a number of advantages over the state of the art systems. First, it allows the video devices to receive an incomplete schedule immediately, and begin executing it even while later events in the playlist are still being processed. By delivering the events that are close to air, it allows the system to go on air sooner than if the entire playlist had to be loaded before any video actions could begin. Second, it allows the video devices to report on the status of events in the playlist even before the download of the playlist is complete, allowing the system to capture a timely record of the video that actually played for purposes such as accounting and fault analysis. Third, it allows the operator interface to remain "live" during the initial download of commands to the video equipment. The operator can determine the status of equipment, view the complete or incomplete playlist, interact with the devices, and request edits to the playlist, even while the initial download is proceeding.

[0006] The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

Fig. 1 is a high level data flow diagram of the throttler, as connected to a broadcast automation system;

Fig. 2 is an illustration of rules for accumulating deletion and insertion commands;

Fig. 3 shows a representation of the data structures used in the throttler;

Fig. 4 is a flow diagram of the method of the throttler's main process;

Fig. 5 is a flow diagram of the method of the throttler's Fill process;

Fig. 6 is a flow diagram of the method of the throttler's Drain process; and

Fig. 7 is a flow diagram of the alternate method of the throttler's Drain process used for urgent commands.

[0007] Referring to the drawings, and more particularly to Fig. 1, the data flow of commands and edits through a preferred embodiment of the throttler is shown. The throttler 100 loads the initial playlist 106 while also accepting edit commands 108. Non-edit commands 114 are received by the throttler 100 and passed directly to the broadcast automation system 118 which typically resides on the same CPU as the throttler, or at least has a device driver on the same CPU as the throttler to allow communication between the two processes. Using the method described below, the throttler 100 interleaves these events and edit commands and generates and modifies the playlist of scheduled events 116. The throttler 100 sends the events to the broadcast automation system 118 for execution which drives the audio and video devices 120 based on the scheduled events. The throttler periodically yields the central processor so that time is available for other processes to handle non-edit commands, such as operator query of the playlist, direct operator command of the devices, and status reporting from the devices. The throttler is best practiced with a broadcast automation system which reads a playlist, as formatted and communicated by the throttler and reformats, if necessary, and then forwards the edit and non-edit commands to a number of audio, video or other device drivers for managing the broadcast automation. The preferred broadcast automation system also displays status of the scheduled events and allows some manual modification by an operator through a user interface.

[0008] For each editing command 108 that has not been processed by the throttler 100, up to two pieces of information are maintained: one deletion command and one insertion command. Either command may be omitted. Each command, or event, has a unique "event identifier."

[0009] When the throttler 100 accepts a deletion command 110, if any prior command applying to the same event identifier, either insertion or deletion, has not been processed, it is discarded, and the newly-accepted deletion command alone is retained. When the throttler 100 accepts an insertion command 112, any previous insertion command that applies to the same event identifier is discarded, but any previous deletion command is retained.

[0010] Fig. 2 illustrates the rules for accumulating deletion and insertion commands. The first column 200 shows the two possibilities for existing insertion and deletion commands for an event scheduled in a playlist. The second column 202 shows the newly accepted command, and the third column 204 shows the resulting command structure for that event. For instance, if event one 206 has no scheduled insertion or deletions and a deletion command 208 is accepted, the resulting scheduled event is a deletion 210 for this event. Event eight 212 has a deletion and an insertion already scheduled. If a new insertion command for this event is accepted 214, then the result 216 is to retain the deletion command and substitute the newly received insertion command and discard the original insertion command. It can be seen by Fig. 2 that the throttler always maintains the minimal set of changes needed to make the events in the automation system correspond with the desired set of events.

[0011] The command pairs 200 and 204, in turn, are organized into a "priority queue" which is a data structure that allows rapid search for the element of the least value. The ordering of the pairs is defined by the scheduled execution times of the events. If there are both deletion and insertion commands, the earlier of the scheduled times of the deleted and inserted copy of the event determines the precedence of the pair. This scheme orders the commands by their relative urgency, while still preserving the fact that the old copy of the event must be deleted before the new one is inserted.

[0012] The priority queue data structure chosen has the attribute that elements of the queue, once inserted, do not change memory location. The fact that memory locations are kept stable allows the hash table to be maintained as a distinct data structure from the priority queue. Were queue elements to change their position in memory, the hash table would have to be updated every time one was moved, necessitating either another search of the table or else maintenance of a pointer to the hash table element inside the priority queue element, and complicating the programs that maintain the queue. The priority queue data structure also allows rapid deletion of an element from any position in the queue. These restrictions mean that a heap, a sorted vector, or a B-tree would be inappropriate data structures. The preferred embodiment uses a "leftist tree," which is a structure well known to those skilled in the art, to organize the priority queue. A more complete description of this data structure may be found in <u>The Art of Computer Programming, Volume 3: Sorting and Searching,</u> by D. E. Knuth (Reading, Mass.: Addison-Wesley 1973 pp. 149-153, 159, 619-620). The leftist tree has the advantage that its performance is faster for operations near the front of the queue. This property makes it preferable to alternative implementations that use AVL trees, splay trees, or similar self-organizing data structures.

[0013] The priority queue is augmented with a hash table, which is also a data structure well known in the art. The hash table maps event identifiers to the address of the queue elements as shown in Fig. 3. This structure is used to locate the delete-insert pair when a new command arrives. Referring to Fig. 3, each Event Identifier 302 has a pointer 304 associated with it that maps by hashing into the queue elements of delete-insert pairs 306.

[0014] The algorithms used in the throttler comprise two processes: "Fill" and "Drain." The Fill process accepts commands rapidly using the method of Figs. 4 and 5. The Drain process mediates delivering commands in a way that allows the broadcast automation system to continue to perform other tasks, such as device control and operator interface, even as new commands are arriving, according to the method of Fig. 6.

[0015] Referring to Fig. 4, the initial load of the playlist reads in the events from the initial playlist 403 in function block 402. If there is another event on the playlist, as determined in decision block 404, then the priority queue and hash table are populated by the Fill process, to be described below, in function block 406. This process continues until all initial events have been loaded into the priority queue. These operations are time inexpensive operations compared with sending the events to the devices, as is done by the broadcast automation system. Once the initial priority queue is constructed, the Fill process awaits commands from its external interface (e.g. other programs, the operator, and the devices) in function block 408.

[0016] Each newly received command is checked to determine whether it is an edit command in decision block 410. If it is not an edit command then it is directed to the correct component of the system and processed in function block 412. Otherwise, the playlist must be edited by adding the new command and updating the priority queue and hash table by calling the Fill command in function block 414.

[0017] Referring to Fig. 5, for each command accepted by the throttler the Fill process first accesses the hash table to find any pre-existing command pair for the event being edited in function block 502. If a pre-existing pair is found in decision block 504, it is removed from the priority queue for processing in function block 506. Otherwise, a new, empty, command pair is created for processing in function block 508. The newly arrived command is then combined with the command pair according to the rules as shown in Fig. 2.

[0018] The command pair is inserted into the priority queue in function block 512, ensuring that it will be ordered correctly according to urgency. Finally, the hash table is updated to reflect the new address of the priority queue entry in function block 514. The Drain process, as described below, is re-enabled in function block 516.

[0019] The Fill process normally takes precedence over the other processes in the system. Because its tasks are only to maintain the hash table and priority queue, it normally consumes only an insignificant fraction of the total central processor unit (CPU) time, and no precautions to keep it from locking out other processes are required.

[0020] The Drain process is usually enabled by the broadcast automation system to retrieve commands at a certain minimum time interval, calculated to leave it enough time for its other tasks. An alternative method would allow commands with less than a specified time to completion to be forced through, even if sending these events to the broadcast automation system would temporarily "freeze" the operator interface, delay the reporting of status of earlier events, postpone the acceptance of non-edit commands, or otherwise temporarily result in undesirable postponement of less urgent tasks. The Drain process consists of an endless loop.

[0021] The Drain process typically communicates with a "device driver" process to control when it is enabled. The control for when it is enabled can be extremely simple; often it is a simple timer interrupt that causes it to be enabled a certain number of milliseconds after processing its last command or a certain number of milliseconds after the device presents a "clear to send" indication. The range of time delays that will result in acceptable performance is normally quite wide. Too short a time delay will overload the CPU and result in undesirable postponement of other processes, while too long a time delay will cause events to reach the devices after their scheduled times, as could happen in the method of Fig. 6, or always be processed as "urgent" events, as in the alternate method of Fig. 7. Normal workloads in a system capable of handling eight channels of video indicate that delays in the range of a few hundred milliseconds to a few seconds all result in acceptable performance.

[0022] Referring now to Fig. 6, the simple Drain process is shown. First, the priority queue is checked to determine whether there are command pairs in the priority queue in decision block 602. If the is queue is empty, then the process is blocked until a command pair arrives in function block 604. The Drain process waits until the Fill process re-enables it, as shown in Fig. 5, function block 516. Otherwise, a check is made to determine whether the automation system is ready to accept a new command in decision block 606. If not, the Drain process is blocked, and is re-enabled when the system is ready to accept more commands.

[0023] When there are events to remove from the priority queue and the system is ready to receive them, the first command pair is retrieved from the queue in function block 610. When a command pair has been retrieved, it is deleted from the priority queue, and its corresponding entry in the hash table is also deleted in function block 612. The command pair is presented to the broadcast automation system in function block 614. Once the command pair has been successfully sent, the process yields the CPU to other processes, in function block 616, to ensure that the command processing process can respond to requests and then continues again in decision block 602 to process additional command pairs from the priority queue.

[0024] An alternate method which ensures timely processing of urgent commands is shown in Fig. 7. This process is similar to the simple Drain process. First, the priority queue is checked to determine whether there are command pairs in the priority queue in decision block 702. If the is queue is empty, then the process is blocked until either a command pair arrives, the automation system becomes ready, or the time interrupt for urgent events occurs in function block 718. Otherwise, if the queue is not empty, the first command pair is retrieved from the queue in function block 704. A check is made to determine whether the automation system is ready to receive a new command in decision block 706. If it is not ready, a test is performed to determine whether the command is urgent in decision block 708. If it is not urgent, then the timer interrupt is set for a time when the first event becomes urgent in function block 710. The Drain process is again blocked as described above in function block 718. If the command is urgent, as determined in decision block 708, or the automation system was ready to receive a command, as determined in decision block 706, the command pair is deleted from the priority queue, and its corresponding entry in the hash table is also deleted in function block 712. The command pair is then presented to the broadcast automation system in function block 714. Once the command pair has been successfully sent, the process yields the CPU to other processes, in function block 716, to ensure that the command processing process can respond to requests and then continues again in decision block 702 to process additional command pairs from the priority queue.

Claims

1. A throttler used for rapid start-up of a broadcast automation system comprising:

   means for loading a playlist;

   means for accepting a plurality of editing and non-editing commands;

means for interleaving said editing commands; and

means for presenting said interleaved editing commands to a broadcast automation system, allowing said broadcast automation system to process non-editing commands with said interleaved editing commands.

2. A throttler as recited in claim 1, wherein said editing commands are either insertion or deletion commands and an event comprises a command pair of no more than one insertion command and no more than one deletion command, each said event having a unique event identifier, and wherein one command in said command pair may be empty.

3. A throttler as recited in claim 2, wherein each said command pair is stored in a rapidly accessible priority queue ordered by urgency of each said event.

4. A throttler as recited in claim 3, wherein a command pair stored in said priority queue is addressable by either an event identifier or as a lead element in said priority queue.

5. A throttler as recited in claim 4, wherein said priority queue allows deletion of a command pair identified by said event identifier which may be located anywhere within said priority queue.

6. A throttler as recited in claim 1, wherein said means for interleaving said editing commands further comprises:

means for "filling" or accepting commands, and

means for "draining" commands by mediating delivery of said accepted commands to a broadcast automation system.

7. A method for throttling commands to be used for rapid start-up of a broadcast automation system, said method comprising the steps:

receiving commands from external interfaces;

determining whether said received commands are of type editing or non-editing commands;

forwarding non-editing commands to a broadcast automation system;

filling and rescheduling a playlist with said editing commands; and

draining said playlist of commands by sending command pairs to said broadcast automation system.

8. A method for throttling commands as recited in claim 7, further comprising the step of inputting an initial playlist.

9. A method for throttling commands as recited in claim 7, wherein said draining step may interrupt said filling step if said command pair is scheduled for immediate execution.

10. A method for throttling commands as recited in claim 7, wherein said filling step or said broadcast automation system enable said draining step when desired.

11. A method for throttling commands as recited in claim 7, wherein said playlist comprises a plurality of events, each said event comprising editing commands which are either insertion or deletion commands and wherein an event comprises a command pair of no more than one insertion command and no more than one deletion command, each said event having a unique event identifier, and wherein one command in said command pair may be empty.

12. A method as recited in claim 11, wherein each said command pair is stored in a rapidly accessible priority queue ordered by urgency of each said event.

13. A method as recited in claim 12, wherein a command pair stored in said priority queue is addressable by either an event identifier or as a lead element in said priority queue.

14. A method as recited in claim 13, wherein said priority queue allows deletion of a command pair identified by said event identifier which may be located anywhere within said priority queue.

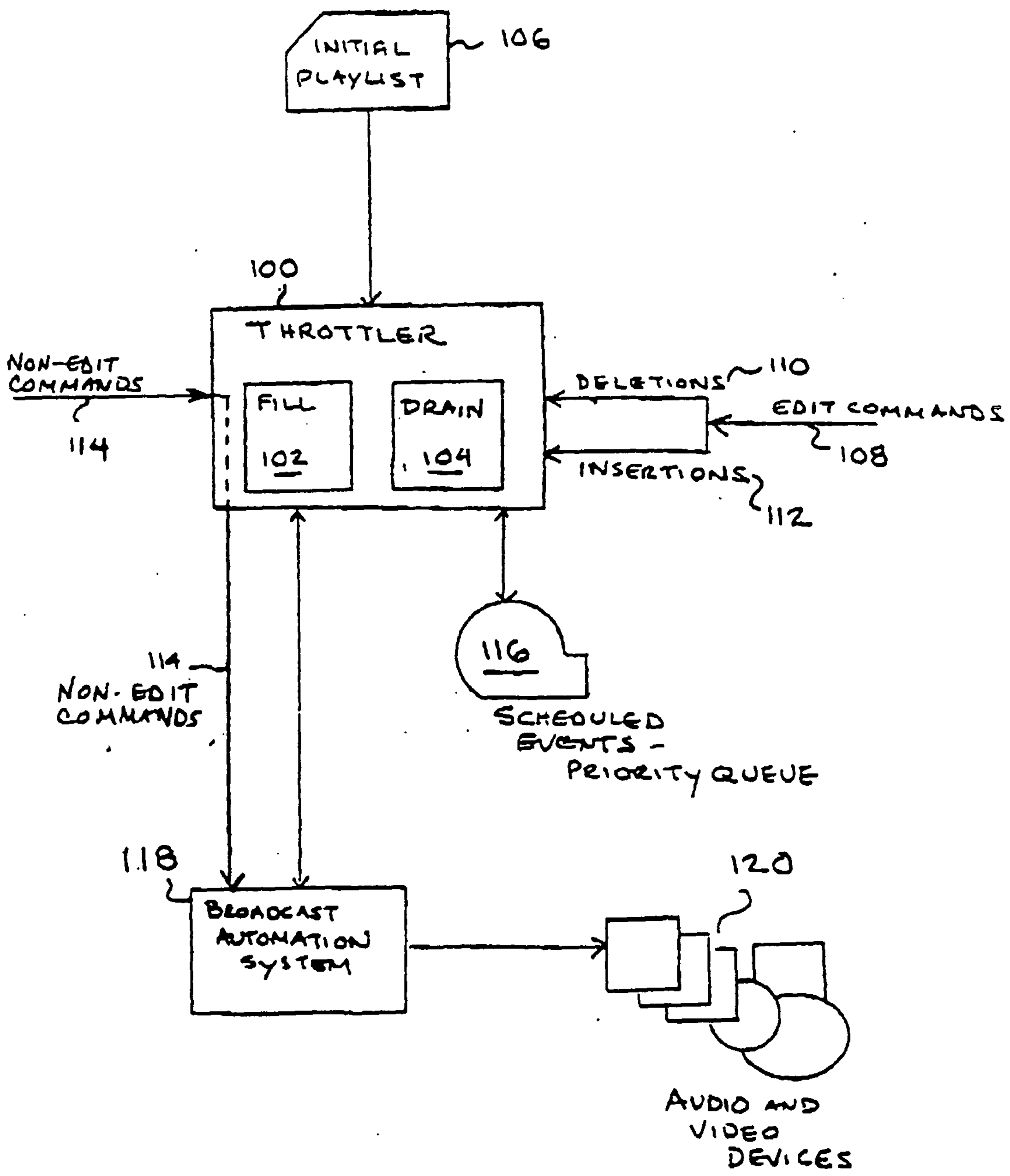

INITIAL PLAYLIST
106
100
THROTTLER
FILL
102
DRAIN
104
NON-EDIT COMMANDS
114
DELETIONS
110
EDIT COMMANDS
108
INSERTIONS
112
116
SCHEDULED EVENTS - PRIORITY QUEUE
114
NON-EDIT COMMANDS
118
BROADCAST AUTOMATION SYSTEM
120
AUDIO AND VIDEO DEVICES

FIG. 1

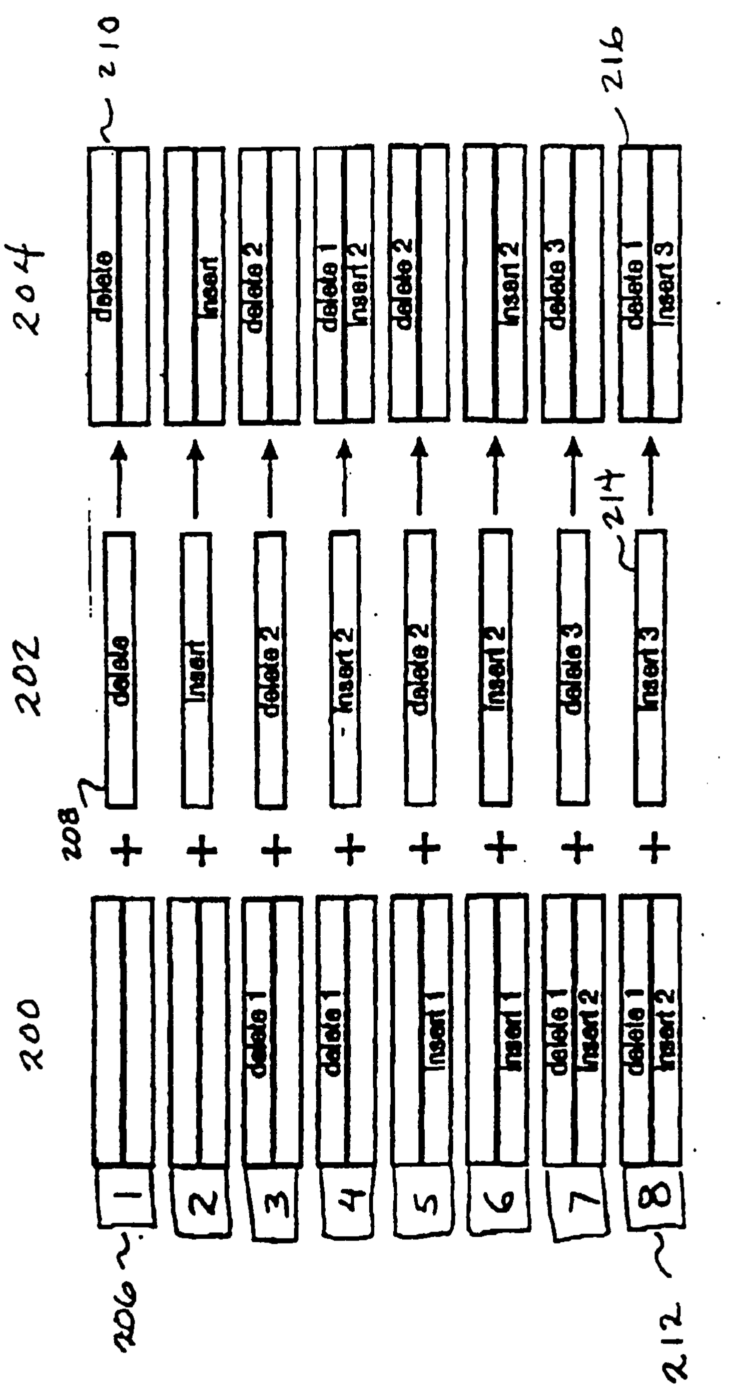

200
202
204
206
208
210
212
214
216
1
2
3
4
5
6
7
8
delete 1
delete 1
insert 1
insert 1
delete 1
insert 2
delete 1
insert 2
delete
insert
delete 2
- insert 2
delete 2
insert 2
delete 3
insert 3
delete
insert
delete 2
delete 1
insert 2
delete 2
insert 2
delete 3
delete 1
insert 3

FIG. 2

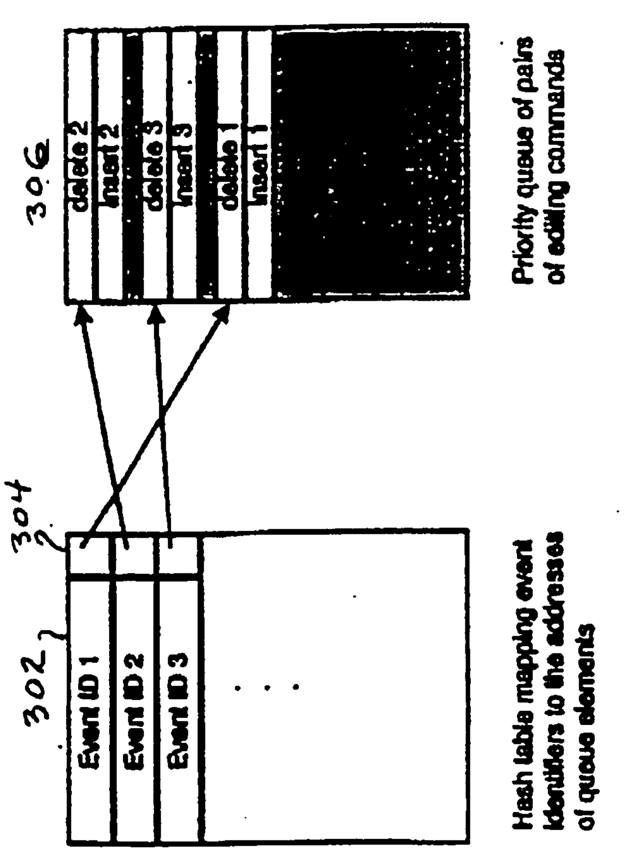

302
304
Event ID 1
Event ID 2
Event ID 3
. . .
Hash table mapping event identifiers to the addresses of queue elements
306
delete 2
insert 2
delete 3
insert 3
delete 1
insert 1
Priority queue of pairs of editing commands

FIG. 3

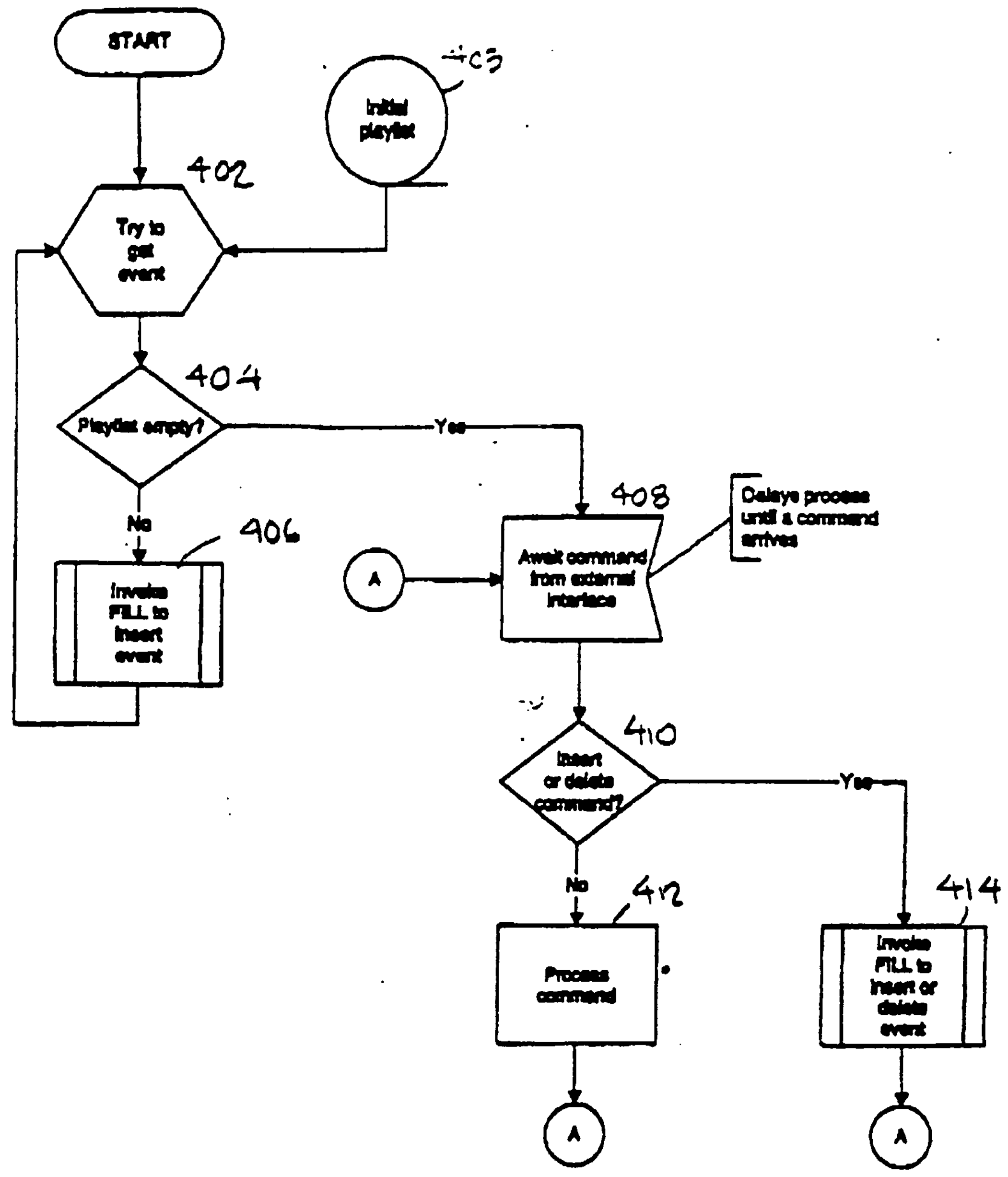

START
Initial playlist
403
402
Try to get event
404
Playlist empty?
Yes
No
406
Invoke FILL to insert event
408
Delays process until a command arrives
A
Await command from external interface
410
Insert or delete command?
Yes
No
412
414
Process command
Invoke FILL to insert or delete event
A
A

FIG. 4

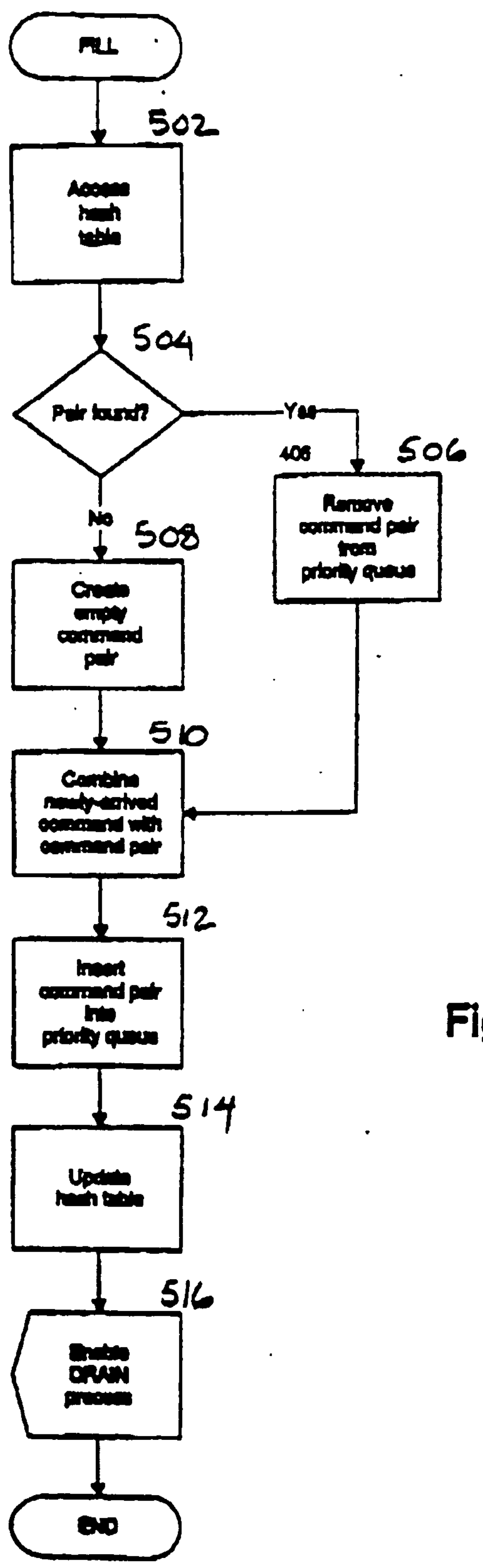

FILL
502
Access hash table
504
Pair found?
Yes
406
506
Remove command pair from priority queue
No
508
Create empty command pair
510
Combine newly-arrived command with command pair
512
Insert command pair into priority queue
514
Update hash table
516
Enable DRAIN process
END

Fig. 5

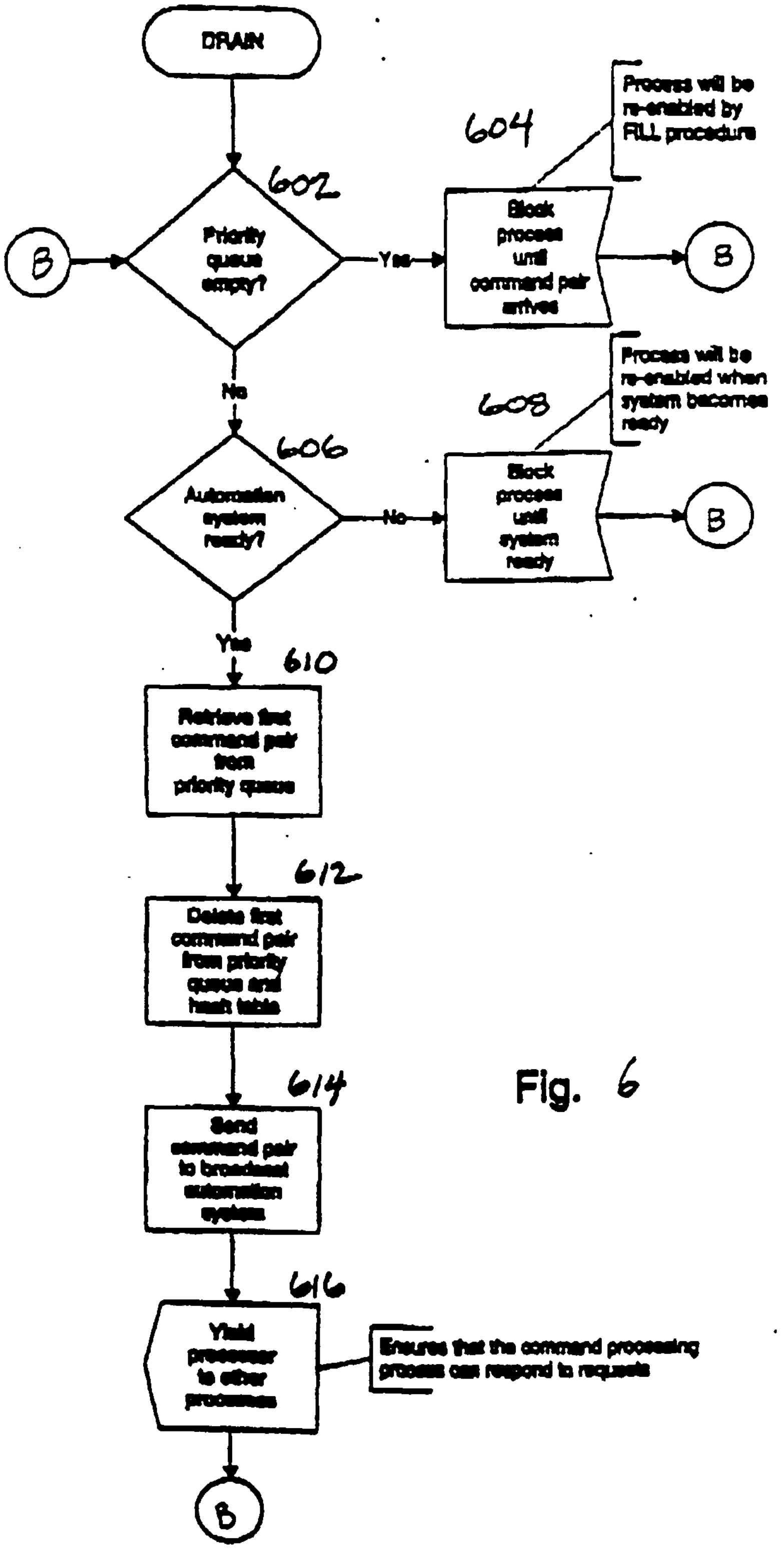
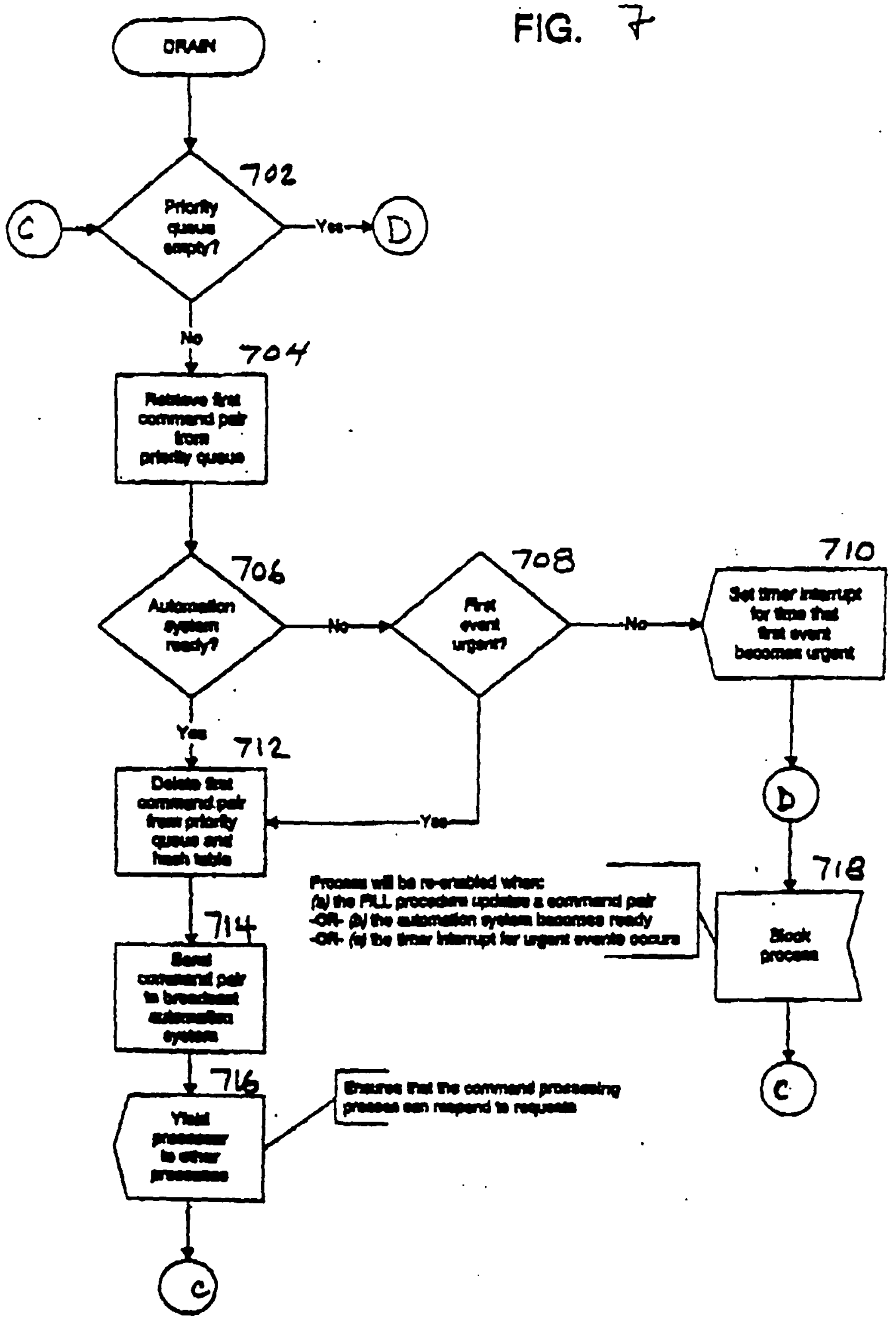

DRAIN
602
Priority queue empty?
Yes
604
Block process until command pair arrives
Process will be re-enabled by FILL procedure
B
B
No
606
Automation system ready?
No
608
Block process until system ready
Process will be re-enabled when system becomes ready
B
Yes
610
Retrieve first command pair from priority queue
612
Delete first command pair from priority queue and hash table
614
Send command pair to broadcast automation system
616
Yield processor to other processes
Ensures that the command processing process can respond to requests
B

Fig. 6

FIG. 7


DRAIN
702
Priority queue empty?
Yes
No
C
D
704
Retrieve first command pair from priority queue
706
Automation system ready?
No
708
First event urgent?
No
710
Set timer interrupt for time that first event becomes urgent
Yes
712
Delete first command pair from priority queue and hash table
Yes
714
Send command pair to broadcast automation system
716
Yield processor to other processes
Ensures that the command processing process can respond to requests
C
D
718
Block process
Process will be re-enabled when:
(a) the FILL procedure updates a command pair
-OR- (b) the automation system becomes ready
-OR- (c) the timer interrupt for urgent events occurs
C